School of Science

CPT222
Software Architecture Design and Implementation

Assessment 1: Single Player Implementation

Assessment Type: Programming assignment

Due date: Sunday of Week 7 11:59PM (AEDT)

Weighting: 25%

Overview

This assignment requires you to implement a single threaded game engine using Java programming language, and a single player Graphical User Interface for a simple casino style dice game that uses two dice per roll.

Assessment Criteria

This assessment will determine your ability to:

1. Perform iterative development from the design stage through to implementation and testing using commonly available tools and provided test harness.
2. Implement software systems utilising Java programming language and core object-oriented concepts such as inheritance, polymorphism, and abstract classes/interfaces;
3. Design and implement custom GUI utilising AWT/Swing libraries and the Model View Controller architecture.

Learning Outcomes

This assessment is relevant to the following Learning Outcomes:

1. Understand the purpose of object-oriented design and where and how to apply object-oriented concepts such as inheritance, polymorphism, abstract classes/interfaces and generics;
2. Understand and apply common object-oriented design patterns such as model view controller, observer, and façade;

3. Understand and use constructs from the Java Collection Framework;
4. Design and implement user interfaces using Java programming language;
5. Be proficient using an integrated development environment (Eclipse) for project management, coding and debugging.

Assessment details

This assignment requires you to implement a single threaded game engine, and a single player user interface for a simple casino style dice game that uses two dice per roll.
The rules of the games are simple:
- The player enters the bet amount in points (represented by an int value).
- The player invokes the “Roll” option, thereby placing a bet and initiating a new game.
- The house (server) then randomly “rolls” the dice for the player.
- Finally, the house “rolls” the dice against the player.
- Highest score (sum of two dice faces) wins. A draw is a no contest and the bet is returned to the player.

For this assignment you will be provided with several interfaces and a custom exception class. You must implement those interfaces to provide the required behaviour. You will also be provided with a basic console client (Client.java) which will help testing your initial game engine implementation independently of your GUI (to help you get started early). Moreover, to further facilitate testing of your solution, a dedicated TestHarness.java class will be used to assess the core functionality of your submission during marking. This will also ensure that all functionality is separated from the GUI since your game code should operate independently using the console-based client.
NOTE: You may create additional sample clients to facilitate testing but must ensure that you do not change any of the provided interfaces, the exception class, the console client, and the TestHarness prior to submission. Any changes to those entities will result in marks deduction.

User Interface Requirements

You are to develop an AWT/Swing user interface that supports the following functionality:

1. Add a player. This includes all the basic player details and initial credit points balance.
2. Allow the player to enter bet amount in points and place a bet (invoking the “Roll” option)
3. Display intermediate “rolls” for a player (dice faces should be shown while the dice are rolling)
4. Display final roll outcome for a player (faces of the “landed” dice and the total score)
5. Display intermediate “rolls” for a house (dice faces should be shown while the dice are rolling)
6. Display final roll outcome for a house (faces of the “landed” dice and the total score)
7. Display current game result for a player including:
- Game result (won/lost/drew), see GameEngine.GameStatus enum;
- Original bet amount
- Updated player points balance
8. Display player’s history (game results) of up to five previous games.
9. Allow the player to top-up (change) the available credit points.

It is up to you how to design the layout and appearance of your interface, and you should focus on clarity, simplicity, and usability rather than elaborate design. However, you should include at least one each of the following:

- A pull down menu;
- An input dialog box and an error notification dialog box (e.g. to deal with insufficient funds);
- A toolbar with buttons for performing all important system functions;
- A panel which represents the rolling dice (this can be as simple as a single text label that is updated for each new pair of dice numbers received from the GameEngineCallback methods - you don't have to use elaborated graphics here).

Implementation Specifics

- You must implement the provided GameEngine, Player, GameEngineCallback, DicePair and Dice interfaces, in classes called GameEngineImpl, SimplePlayer, GameEngineCallbackImpl, DicePairImpl, and DiceImpl. You must provide the behaviour specified by the provided source code comments. The imports in Client.java and TestHarness.java show you which packages these classes should be in.

- More specifically, you must provide appropriate constructors (these can be determined from Client.java/TestHarness.java) and method implementations (from the five interfaces) in order to ensure that your solution can be complied and tested without modifying the Client.java/TestHarness.java. A sample output trace will be provided to demonstrate the expected/correct system behaviour; you do not need to follow the exact output format.

- All of your GUI code (MVC view(s) and controllers(s)) should be separate from, and use, the to be implemented GameEngineImpl class (i.e. MVC model) as exposed via the GameEngine façade[1] interface.

- You should use assertions to test pre- and post- conditions where applicable. e.g. the provided bet amount cannot be negative.

- For Assignment 1 you need only provide a single player/single-threaded[2] implementation of the game engine. In Assignment 2, you will extend this code to add: multi-threading, multiple player windows, and networking capability to create a proper concurrent and distributed multi player game.

NOTE: The provided interfaces are designed for multiple players (to support Assignment 2) and the methods should be implemented accordingly, but your GUI in Assignment 1 need only support a single player.

Other important points:
- You should aim to provide high cohesion and low coupling.
- You should comment all important sections of your code.

Referencing guidelines

You must acknowledge all the courses of information you have used in your assessments. You are free to refer to textbooks and notes and discuss the design issues (and associated general solutions) with your fellow students on the Discussion forum; however, the assignment must be your own individual work. Where you do make use of other references, please cite them in your work. Note that you will only be assessed on your own work, so the use of third party code or packages is not advised.

Submission format

The source code for this assignment (i.e. a complete compiled Eclipse project) should be submitted in a form of a zip file through the *Assignments* section of Canvas. You can develop your system using any IDE, but will have to create an Eclipse project using your source code files for submission and assessment purposes. Upload your Eclipse project as one (1) single zip file. Make sure the file type is an accepted format (.zip, .7Z).

Academic integrity and plagiarism

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own solutions. If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct.

Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to the University website.

Assessment declaration

When you submit work electronically, you agree to the assessment declaration (links to an external site).